December 1990 THESIS/DISSERTATION

An Implementation of Opportunistic Scheduling for Robotic Assembly

Allan Wayne Butler

AFIT Student Attending: Texas A&M University

AFIT/CI/CIA-90-125

AFIT/CI
Wright-Patterson AFB OH 45433-6583

AN IMPLEMENTATION OF OPPORTUNISTIC SCHEDULING FOR ROBOTIC ASSEMBLY

A Thesis

by

ALLAN WAYNE BUTLER

Submitted to the Office of Graduate Studies of
Texas A&M University
in partial fulfillment of the requirements for the degree of

MASTER OF SCIENCE

DECEMBER 1990

Major Subject: Industrial Engineering

ABSTRACT

An Implementation of Opportunistic Scheduling for Robotic Assembly. (December 1990)

Allan Wayne Butler

B.E., Brigham Young University

Chair of Advisory Committee: Dr. Cesar O. Malave

The goal of this ~~research~~ thesis is to combine computerized vision and artificial intelligence programming in an application of robotic assembly that will use opportunistic scheduling. Opportunistic scheduling is making a schedule based on current "opportunities". A robot provided with a vision system has the capability of *recognizing* random opportunistic events. However, vision systems have many limitations. A heuristic method of taking pictures is developed to improve object recognition reliability. The robot is given basic assembly knowledge using the production rule methodology, and assembly precedence information using a database of partial order sets. Dynamic state information is also maintained by the program. Parts are delivered randomly on conveyor belts. The robot is given the capability of assembling a mix of products and assembling multiple products concurrently. Thus, the robot can assemble a product in any feasible way and schedule an optimal plan according to the random arrival of parts. A user friendly interface with the robot is developed.

ACKNOWLEDGEMENTS

I would like to thank my committee for the counseling and guidance they gave to me during my research. A special thanks to Dr. Cesar O. Malave, my committee chairman, who spent a great deal of time helping me to develop my ideas, and who encouraged me to make my non-thesis project into a full thesis.

My greatest appreciation goes to my wife Bev and to my children Michael, David, and Britta for their unselfish love and undying support throughout the eighteen months we have been at Texas A&M.

TABLE OF CONTENTS

TABLE OF CONTENTS (Continued)

LIST OF FIGURES

LIST OF TABLES

CHAPTER I

INTRODUCTION

Methods of assembly have come a long way since the first assembly lines of Henry Ford. F. W. Taylor, for example, using scientific management in his studies, was the first to recommend short breaks for the workers as a way to reduce the effects of fatigue and improve productivity [1].

The manual assembly lines of today have been improved by using new techniques for line balancing, frequently changing worker assignments, and allowing workers to control the speed of the conveyor system [2]. Even with these advancements, many new kinds of problems have resulted from our continuing technological revolution. The requirement for smaller tolerances and higher quality generates a need for increased consistency and standardization in manufacturing processes. In addition, many tasks which are strenuous or even hazardous are still being done by humans.

1.1 Trends in the Market Place

Referring to the available colors of the mass produced Model-T, Henry Ford said, "They can have any color they want as long as it's black". This humorous comment was indicative

---

This thesis follows the style and format of the *IEEE Transactions on Robotics & Automation.*

of the attitude behind the early methods of mass production.

Today, consumers are not so easily satisfied. Customers want products to be personalized or unique. These types of demands can be easily seen in the automotive industry. Each customer demands unique requirements for color, stereo, air conditioning, cruise control, etc.

Personalized items can not be mass produced. This fact gives rise to batch production, but yet the desire for higher manufacturing flexibility is quickly becoming a requirement. Flexibility is the ability to change from manufacturing one product to a different product with little time lag. Ideally, the most flexible job shop could produce one item, followed by the production of a completely different item using the same machines and without any additional time for changes in set-up. The key here is to minimize set-up time.

As the need for greater flexibility in manufacturing grew, industry returned to manual labor. Man's ability to make almost instantaneous changes in behavior gives him a significant advantage in the constantly changing world of flexible manufacturing. However, the advantages of manual labor are being outweighed by a great disadvantage - the high cost of labor.

1.2 Robots in Assembly

Robots are an extension of computers. As computers have become more powerful, robots have become more capable. At

first, it was hoped that robots would be the solution to many problems in manufacturing, but it is not easy to build a machine as adaptable as humans. Only recently have robots been built that can make significant contributions.

The first large scale use of computerized robots was in 1978 by General Motors as part of their Programmable Universal Machine for Assembly (PUMA) system that included conveyors, part feeders, and assembly robots working alongside humans [3]. Robots are now being used for welding, spray painting, inventory control, material transfer, machine part loading/unloading, and much more [4,5].

It is difficult for robots to adequately perform assembly operations. This difficulty is mainly due to the high degree of dexterity and sensory feedback needed to perform assembly operations. These types of operations are simple for a human, but to an insensitive, inflexible machine they are difficult, at best. Measures of robotic attributes such as work volume, load-carrying capacity, accuracy, repeatability, and arm speed were developed as a method of comparing robot capabilities to task requirements [4].

Robots are the most flexible machine available today. However, there is a great deal of inflexibility that comes with this flexibility. Robots are generally accompanied by many special fixtures to hold work pieces. Special pre-processing has to be done in order to sequence and orient parts, and specialized feeders are required to position parts. There is also the problem of how to deal with errors

(an error occurs whenever any part is slightly misplaced).

In the past, a great deal of time and money was spent designing and building fixtures, forcing the changing world of assembly into the structured environment of the robot. In addition, long hours were spent attempting to program the robot to do many detailed tasks. It has been reported that about 90 percent of the robotic programmed instructions are detailed procedures on how to recover from errors [6].

A great deal of research has been done to find better ways to send information to robots about their surroundings [4]. Tactile sensors indicate when (touch) and how hard (force) the robot touches an object. Range sensors indicate know how close the robot is to an object. Interlock sensors such as microswitches and infrared sensors signal to the robot the presence or absence of an object. Probably the sensor that stimulated growth of robotics the most is computerized vision [4,5].

Despite the high capital investment required and the difficulties discussed above, robotic applications have become cost effective. Robots are now capable of faster assembly times and increased consistency, leading to better quality. Yet, much more is needed. Robots are going to have to get "smarter".

1.3 The Problem

Manual assembly is too costly and robotic assembly is

too inflexible. The problem, then, is that robots must become more flexible in assembly operations to adequately fulfill the production requirements and control the rising prices associated with assembly. This work uses computerized vision, artificial intelligence, and opportunistic scheduling in an attempt to increase robotic flexibility.

1.3.1 Robotic Programming

The program for a robot is essentially a plan. Normally, the programmer has enough information to develop this plan. However, in assembly, there is usually more than one way to assemble a part. The programmer decides which of these many ways the robot will use to assemble the part. The robot will then assemble the part in the same way every time, and the environment must be changed to accommodate the fixed plan.

Research has been done to find methods for automatically generating an optimal assembly plan [7,8,9,10]. Even if an optimal assembly plan was determined using any one of these methods, the optimal solution would change if the arrival sequence of components to be inserted changed over time.

1.3.2 Random Assembly

In today's world of manufacturing the acronym CAD/CAM (Computer Aided Design/Computer Aided Manufacturing) has

caused a great deal of excitement, and significant attempts are being made to implement its concepts. Just-in-Time (JIT) manufacturing is another exciting subject of the day. For simplicity, JIT manufacturing is defined as a method of manufacturing that is directly driven by demand. If a quantity of five is requested by a customer, a quantity of five is produced. When a variety of products are offered, an element of randomness is introduced.

CAD/CAM methods include techniques for generating an aggregate production plan which is used to determine a master schedule for production. The master schedule is a list of quantities for each assembly to be produced in the next planning period. The Goal-Chasing Method [11] is a technique for determining an optimal assembly sequence given a master schedule. Since a customer's request for any particular item is a random variable, the master schedule will continually change, consequently, so does the optimal assembly sequence.

Another part of CAD/CAM is the materials requirement planning (MRP) which controls the ordering, sequencing, and timing of component delivery. The MRP is dependent on the master schedule previously determined. Since the master schedule frequently changes, so does the MRP [12]. Desiring to eliminate inventories (at least maintain them at the lowest possible level) production will correspond very closely to the random demands of the customer. Thus, a fixed optimal production rate and sequence are unacceptable under these dynamic conditions.

1.4 Method of Approach

The goal of this research is to combine computerized vision and artificial intelligence programming in an implementation of opportunistic scheduling that will increase robotic flexibility in assembly. In this section computerized vision and artificial intelligent programming are introduced, and then a literature survey of opportunistic scheduling is presented.

1.4.1 Computer Vision

Computer vision digitizes an image and, through various algorithms, calculates geometric characteristics such as area in square pixels, centroid, and boundary information about the object in view. (Note that a pixel is a single dot of light which composes a picture.) Once the vision system is trained by showing it examples of an object (a prototype), it can match this prototype with a picture of the same object in any orientation (object recognition) [4,5].

A robot provided with a vision system, has a method of identifying placement errors, out of tolerance parts, and improperly sequenced parts. However, there are many limitations to vision systems. Most vision systems can give only two dimensional information (the X and Y axis and the rotation around the Z axis). Another limitation is lighting. The amount and type of lighting used is very important, and

changing either will seriously effect the ability of the vision system to recognize trained objects. (Often, if the lighting is changed, the vision system must be re-trained to recognize the objects under the new conditions). Due to the variability in lighting, each picture taken of an object will be slightly different. Also, the algorithms used for object recognition are very computationally intensive, allowing errors. These types of errors can be due to the camera resolution and the less than perfect matching of images to prototypes. The combination of these two variables produces a significant possibility of an error in object recognition. A heuristic method of taking pictures is developed to improve the overall recognition capability of the vision system.

Object recognition can be a very slow process, especially if objects are allowed to overlap or touch. The assumption in this work is that objects do not overlap or touch. In this way, the vision system simply has to recognize objects under the same conditions in which it was trained.

1.4.2 Artificial Intelligence

As a method of representing sequencing constraints, engineers frequently use precedence diagrams [2,13,17,18]. One good way to program precedence diagrams into the computer is by using partial order representation of knowledge (explained in detail in chapter 3) [15]. This database of

assembly precedence information will be easily maintainable. A user friendly interface with the robot is developed to simplify making changes associated with the dynamic nature of assembly.

The robot is given some basic knowledge about assembly in the form of rules. The robot continually checks the rules until one applies and then executes the procedure that accompanies that rule. This method of programming is called the production rule method. Rules govern such things as when to get a component from the machine buffer rather than look for a new one on the incoming conveyor belt. It also maintains information regarding which components are already inserted and which components can be inserted (state information).

1.4.3 Literature Survey

Opportunistic scheduling is making a schedule based on the current "opportunities" or environmental status [16]. By using the production rule methodology, opportunistic scheduling can be incorporated in robotic programming [15].

The problem arises when the actual arrival rate or the specific sequence of components is different from the fixed conditions programmed into the robot. This is known as the difference between planning and scheduling [13] (or planning and control [14]).

Planning is defined as:

> Given a task and detailed information about present capabilities, select an optimal way to accomplish the task in the future [13].

Scheduling is defined as:

> Given a plan, and detailed information about the present abilities, complete the task in the most robust and time-efficient way possible [13].

In most robotic programs a schedule is given to the robot along with the plan. Yet, the information required to schedule the events previously planned is not actually available until the time of execution. To illustrate this difference, consider the following example. A particular manufacturer has a robotic workstation dedicated to the final assembly of one of his products. The robot has been programmed to assemble the product in a fixed, but optimal way. Components for assembly are delivered to prescribed locations by specialized part feeders, and base assemblies are loaded manually by a technician. As long as tolerances, location, orientation, sequence, and arrival times exactly correspond to the plan and schedule programmed in the robot, production will run smoothly.

If unplanned events occur, however, production will stop. For example, if the robot is due for regular maintenance, assembly of the product would come to a complete stop until work was finished. If a placement error occurred, production would stop until the error was corrected by the human technician. If a component was fed to the robot out of

sequence, assembly would completely stop until the error was corrected.

Similarly, if unscheduled events occur production will stop. If a component does not arrive when the robot is scheduled to pick it up, an error occurs. Thus, a programmed robotic assembly schedule only adds constraints to the environment by forcing it to further conform.

In this research components are delivered to the robot in a partially or completely random order on a simple conveyor belt. This randomness generates the need for a buffer to temporarily store components not yet needed. There are many ways to use a buffer. Probably the first strategy used was one called *fixed-buffering* [13,19]. Using this strategy, the robot takes the random components, identifies them, and places each of them in a known buffer location. Once all the components are located and placed in the buffer, the robot assembles the product using the single fixed plan.

Another strategy is called *fixed-build* [13,20]. This strategy is applicable when components arrive in a bin that contains all the necessary components for the assembly. The robot identifies the components and picks them up in the sequence it needs for its fixed assembly plan.

Both of these strategies are inferior to a strategy that uses opportunistic reasoning [13]. An opportunistic robot is capable of performing an assembly according to any one of the many feasible assembly plans. It can also determine which plan would be optimal according to the sequence in which the

parts are delivered.

If the opportunistic robot was given the capability of assembling a mix of products and assembles multiple products concurrently, an even greater advantage is gained. The advantage of assembling multiple products concurrently is the reduction in the average number of components in the buffer that results from random delivery. For example, when assembling four products, the robot will be looking for at least four different components instead of just one. With the introduction of multiple assemblies, priorities must be used in order to overcome the conflict that occurs when two or more assemblies require the same component at the same time.

1.5 Research Objectives

By giving the robot the ability to "sense" its surroundings and some "intelligence" with regard to assembly, the need for fixtures is significantly reduced and the robot can use opportunistic scheduling in real time production. A robot is interfaced with a computer vision system. The camera for the vision system is mounted on the robot arm so the camera can be positioned by moving the robot arm.

The robot program uses a database of component precedence information and includes rules for assembly. With this knowledge, the robot can act for itself almost "intelligently" as it assembles products according to the

opportunistic schedule it develops at the time of program execution. The objectives for this research are:

1. A user friendly interface with the robot for flexible production of multiple assemblies.

2. A heuristic method of taking pictures to improve overall reliability of object recognition for the vision system.

3. A program that uses a partial order representation of assembly precedence information, and rule-based assembly knowledge.

4. Implementation of the above interface, heuristic search, database, and rule-based program on the Adept™ single arm robot with arm-mounted camera.

1.6 Thesis Outline

Chapters two and three explain the apparatus and approach used. Chapter four gives a discussion of the results, conclusions and recommendations for further research.

CHAPTER II

THE APPARATUS

The task of solving this research problem was simplified by using the "blocks world", and requiring only simple insertion operations. There are twenty wooden blocks to be recognized. Ten of the blocks have simple geometric shapes milled into them. The other ten blocks are cut into the shapes that fit the holes. All the blocks are $^{3}/_{4}$ of an inch thick. The shaped blocks average about one and one half inches across. They were cut out on a band saw and have slightly rounded corners to better fit the holes. The robots work table is black, therefore, the shapes are painted white to give the best possible contrast and camera resolution. The blocks with holes are about two and a half inches square and have the shaped hole generally located toward the center. The holes were milled into the bases with an NC milling machine. Most of the blocks allow about $^{1}/_{16}$th of an inch tolerance between the hole and the shape. The blocks are painted black; and the holes, white. This coloring is to accentuate the hole parameters and hide the overall block dimensions.

The geometric shapes used were: circle, section, triangle, square, pentagon, hexagon, trapezoid, cross, oval, and star (See Table 1). These blocks are connected together to simulate four different products. The circle, section,

triangle, and square are connected to make assembly number 4, and is given component precedence that simulates two plates with two fasteners. The pentagon, hexagon, and trapezoid are connected together to make assembly number 3. The cross and oval make assembly number 2, and the star is assembly number 1.

Table 1. Statistical characteristics of vision prototypes.

Prototype Name	Number Examples Taught	Memory Req.	Physical Dimensions* (XxYxZ) in.	Mean Area (Pixels)	Difference in Area (Pixels)
Circle	10	2	$1^{19}/_{32}x1^{19}/_{32}x^{23}/_{32}$	8365	
Cir.base	10	2	$1^{20}/_{32}x1^{20}/_{32}x^{11}/_{32}$	9251	886
Section	11	4	$1^{16}/_{32}x1^{14}/_{32}x^{23}/_{32}$	7362	
Sec.base	10	4	$1^{18}/_{32}x1^{17}/_{32}x^{11}/_{32}$	8510	1148
Triangle	10	4	$1^{23}/_{32}x1^{16}/_{32}x^{23}/_{32}$	5865	
Tri.base	10	4	$1^{26}/_{32}x1^{19}/_{32}x^{15}/_{32}$	6866	1001
Square	7	4	$1\ ^{9}/_{32}x1\ ^{9}/_{32}x^{23}/_{32}$	6883	
Sq.base	7	4	$1^{11}/_{32}x1^{11}/_{32}x^{15}/_{32}$	7781	898
Pentagon	7	5	$1^{18}/_{32}x1^{16}/_{32}x^{23}/_{32}$	7024	
Pen.base	7	5	$1^{20}/_{32}x1^{18}/_{32}x^{12}/_{32}$	7942	918
Hexagon	7	4	$1^{19}/_{32}x1^{14}/_{32}x^{23}/_{32}$	7295	
Hex.base	7	4	$1^{22}/_{32}x1^{16}/_{32}x^{12}/_{32}$	8404	1109
Trapezoid	7	4	$1^{17}/_{32}x1^{10}/_{32}x^{23}/_{32}$	6943	
Tra.base	7	4	$1^{21}/_{32}x1^{12}/_{32}x^{17}/_{32}$	7931	988
Cross	7	7	$1^{21}/_{32}x1^{22}/_{32}x^{23}/_{32}$	7013	
Cro.base	7	7	$1^{24}/_{32}x1^{24}/_{32}x^{12}/_{32}$	8391	1378
Oval	8	3	$1^{11}/_{32}x\ \ 2\ \ x^{23}/_{32}$	8907	
O.base	8	3	$1^{14}/_{32}x2\ ^{4}/_{32}x^{12}/_{32}$	10109	1202
Star	7	6	$2\ \ x1^{30}/_{32}x^{23}/_{32}$	6675	
St.base	7	6	$2^{10}/_{32}x2\ ^{3}/_{32}x^{13}/_{32}$	9140	2465

* All bases are $2^{1}/_{2}x2^{1}/_{2}x^{23}/_{32}$ inches. The dimensions shown are of the matching hole.

2.1 The Robot

The robot is an Adept™ robot which is a SCARA (Selective Compliance Assembly Robotic Arm) type robot. The work volume

for a SCARA type robot is cylindrical with the robot at the center of the cylinder. It has one arm with four degrees of freedom (X, Y, Z, and rotation around the Z axis). The gripper is a small pneumatic suction cup.

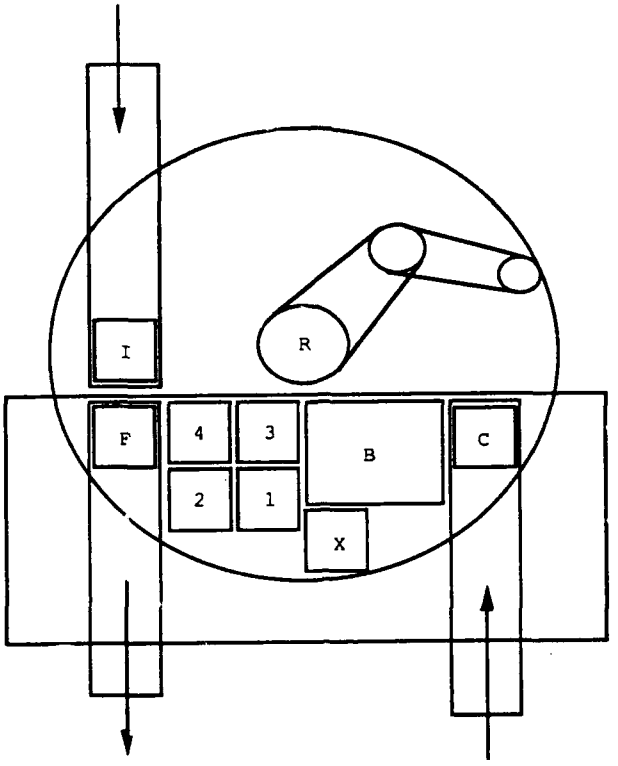
The robot is required to assemble four different assemblies concurrently. Depending on product priorities and amount of work in process, the robot schedules the assembly operations. Through a system of conveyors and infrared sensors, the robot maintains continuous production.

The Adept™ robot uses VALII as its programming language. This application is written in VALII and utilizes the supplemental AdeptvisionII™ commands to operate the associated vision system.

2.2 The Robot Cell

The components to be inserted arrive in random sequence on a conveyor passing through the robots work space (See Figure 1). An infrared sensor stops the conveyor when a component is in the pickup region, and signals to the robot that a component has arrived. The component is randomly oriented and randomly located within the pickup region.

If a needed component is not already stored in the buffer and if a component is present in the pickup region, the robot will go to the conveyor. Using the vision system, the robot will identify and locate the new component. Once the component is identified, the robot will either insert it



R = The robot manipulator and circular work space.
I = Incoming base assembly conveyor and pickup region.
C = Incoming component conveyor and pickup region.
F = Outgoing finished assembly conveyor and drop region.
B = Buffer region for components.
X = Region for placement of rejected parts.
1, 2, 3, and 4 = Work regions by number.

Figure 1. Top view of robotic work cell.

or store it in a buffer location for later use.

Once an assembly is complete, the robot places it on an outbound conveyor for finished products, and then moves to an incoming conveyor to obtain a new base assembly to assemble. Like the components, the base assemblies can be randomly sequenced, randomly oriented, and randomly located within the pickup region of the conveyor. The base assembly is placed

in an empty work region and the state information for that region is updated. (A work region is an area of the robot work space that is designated as a location for assembling products. See Figure 1.)

The buffer has a maximum capacity of ten components, but is a revolving buffer in which the first location is always filled first, then the second, etc. For this reason, buffer location ten is rarely used. However, if the buffer is full, and the robot needs to store one more component, the robot will stop working and signal for help from the operator.

The reject location is where the robot places all the parts (components and base assemblies) that are considered out of tolerance. In this project, the robot is given four work regions, but the number of regions can be adjusted to fit production needs and the size of the products being assembled.

2.3 The Vision System

The camera for the vision system is mounted on the arm of the robot. It produces a grey scale image. Each pixel in the camera image is assigned a numerical value from 0 to 256 (0 being black and 256 being white). The image analyzer is binary, so the gray scaled image must be converted to ones and zeros. This is accomplished by a user defined threshold. All pixel values less than the threshold are converted to black (zero), and all pixel values above the threshold are

converted to white (one).

The camera is mounted on the robot arm in such a way as to allow it to move only in the X-Y plane. Objects up to eight inches outside the robot work space can be seen and recognized by the vision system, but not picked up by the robot arm. As a result of this constraint, all visual searches are limited to the robot's work space. The vision system used by the Adept™ robot is the AdeptvisionII™ system. This system has its own software for training and object recognition. The AdeptvisionII™ system has many parameters that effect object recognition.

In addition to these parameters, AdeptvisionII™ also has many programming commands used for incorporating vision analysis into regular robotic programming. The commands used in this research are listed in Table 2 with a brief interpretation of the function of each [21].

2.3.1 More Inflexibility

Even though the vision system gives the robot great flexibility, it, in and of itself, is very inflexible. The lighting, camera focus, and camera aperture setting all must be held constant. If an object is moved closer to (or farther from) the camera (or if the camera is moved closer to or farther from the object), the vision system will not recognize the object. This peculiarity occurs because object "appears" larger (or smaller) than the prototype given to

the vision system.

Reflections and shadows on or around the objects cause problems with recognition. Standard incandescent light bulbs produce sharp reflections and shadows, where as florescent light sources produce diffuse light with soft shadows.

Table 2. Vision system program commands

COMMAND	DESCRIPTION
VPICTURE	- Directs the vision system to take a picture and hold it for later analysis.
VLOCATE	- Executes vision analysis of a picture previously taken.
VFEATURE	- Command used to access the information resulting from vision analysis.
VQUEUE	- Chronological list of objects seen and their respective VFEATURE information.
VTRAIN	- Command used to initiate a training session to teach prototypes to the vision system.
VLOAD	- Command used to load a vision prototype into the active vision memory.
VSTORE	- Command used to store vision prototypes on disk after a training session.
VDISPLAY	- Command for changing the mode of image display on the monitor.

Room lighting was used along with a florescent ring light located around the lens of the camera. Since the room is large, lighting consisted of 20 panels, each with four florescent lamps. The ring light around the camera is focused at the center of the picture frame. The difficulty with shadowing was considerably reduced by using a transformation to place the object in the center of the camera where the advantage of the ring light was the greatest (This topic will be discussed later in this chapter).

2.3.2 Determining Vision Parameters

When setting vision parameters, the overall goal is to minimize the time spent by the robot in identifying and locating objects. Table 3 is a list of the parameters used and a short interpretation of their effects [21].

Table 3. Vision system parameters

PARAMETER	DEFAULT	APPLIED	DESCRIPTION
V.BORDER.DIST	0	0	Reduce image border to mask out clipped objects.
V.FIRST.COL	1	1 - 150	Set first column of camera data to be processed.
V.FIRST.LINE	1	1 - 150	Set first line of camera data to be processed.
V.LAST.COL	375	250 - 375	Set last column of camera data to be processed.
V.LAST.LINE	483	350 - 483	Set the last line of camera data to be processed.
V.MAX.AREA	262144	15000	Set size of largest object to process.
V.MAX.PIXEL.VAR	1.5	1.5	Set max pixel deviation for fitting image boundary.
V.MAX.TIME	10.0	1.1	Set max time for recognition analysis.
V.MAX.VER.DIST	3.0	1.9	Set pixel tolerance for match verification.
V.MIN.AREA	16	2900	Set size of smallest object to process.
V.MIN.HOLE.AREA	8	8	Set size of smallest hole to process. (not used)
V.THRESHOLD	127	110	Set grey scale value that separates black from white.

The parameter V.MAX.AREA controls the size (in square pixels) of the largest object that the vision system will

attempt to process. The default value for this variable is 262,144 square pixels. Reducing the size of the largest area to be processed will reduce vision system processing time by eliminating the need to process large images like fixtures. In this application, the maximum size of an object to be analyzed was arbitrarily set at approximately 1.5 times the area of the largest part to be identified, the base assembly for the oval. Observations indicate that the vision system gives an average area of 10,109 square pixels to the oval base. Thus, the parameter V.MAX.AREA was set to 15,000.

Similarly, the parameter V.MIN.AREA controls the size (in square pixels) of the smallest area to be processed. Increasing this parameter can reduce processing time by eliminating small, insignificant objects and "ghost" images due to reflections and other imperfections in the work pieces. This parameter was arbitrarily set by taking approximately one half the area of the smallest part. The smallest object in this application was the triangle at 5,865 square pixels. Thus, the parameter V.MIN.AREA equals 2,900.

Upon setting the lighting as desired, a picture was taken of one of the objects, and the threshold was adjusted until a clear, sharp image was displayed on the monitor. Using this threshold value, all the objects were examined to produce clear images. If an image was not clear, adjustments were made. Once the vision system consistently produced clear, sharp images for all the parts, the camera was calibrated, and a transformation matrix was calculated to

transform camera image locations into robot world locations. The threshold value used was 110.

One of the most important variables, other than threshold and lighting, was the tolerance used to confirm the recognition of an object. This parameter is dependant on many things. Since the tolerance is measured in pixels, the length of a pixel is important. To determine this value for the vision system, the square component was used. The vision system reported the area of the square to be 6883 square pixels. Careful measurement revealed that it was 1 $^{9}/_{32}$ inches on a side. Using this information, one inch equals 64.75 pixels, or a ratio of approximately 1:65 (accounting for rounding of the corners etc.). This relationship produces a vision system resolution of 0.015 inches, the smallest distance the vision system can measure.

The resolution of a vision system can be improved by moving the camera closer to an object (which also requires re-training of prototypes), or by exchanging the camera itself for one with a longer focal length (i.e. from a 25mm camera to a 50mm camera). Upgrading the vision board can also improve resolution, but a high resolution vision board is being used in this research.

Quality control checks can be done by setting the number of pixels used to confirm object recognition. For example, using the 0.015 inch resolution, a tolerance of plus or minus 0.030 inches on a part would require a tolerance of two pixels for the vision system. This allows an image to be two

pixels too big or too small and still be matched with the prototype.

In an actual manufacturing setting, this tolerance would correspond to the design tolerances of the part being produced. In this research, however, this value was determined to be the largest value possible without misidentifying a part. Of the shapes and holes used, the set with the smallest difference in pixel areas were the circle and the circle base. Careful measurement showed that the circle had a diameter of $1\ ^{19}/_{32}$ inches and the circle base (the hole) had a diameter of $1\ ^{21}/_{32}$ inches, a difference of 0.0625 inches in diameter. From the resolution determined above, 0.0625 inches is equivalent to 4.0 pixels. The tolerance for the diameters should, then, be less than 2.0 pixels. The parameter V.MAX.VER.DIST controls this variable, and is set to be 1.90.

2.4 Prototype Training

Giving the vision system prototype images for later matching (training or teaching) is a critical process, and many factors must be considered before beginning. Once all the vision parameters and lighting conditions have been decided, it is a good idea to learn how the vision system will represent the boundaries of the part [22]. In most cases the vision representations are not what the user envisions. Knowing how the vision system models the boundary

of the part will help in editing the prototype boundaries. For example, if the vision system always represents a curved surface with a line, then making that surface a line in the prototype will speed up vision recognition.

The vision system needs multiple examples (about 10) of each part being trained. The first example is the most important, because it is the model for all others. This means that all other examples must have the same number of corners and the same type of edges.

The vision system could be trained to find pre-determined gripper points for the objects, but the top surfaces of all blocks used in this research are flat. So, the robot simply picks them up by using the centroid of the shape determined by the vision system.

Correctly orienting the component before placing it in the hole was one of the more difficult functions to program. This function can actually be done in many ways, but the easiest, and the method used, was to simply teach the component and the hole to the vision system in the exact same orientation. The reason this method works is because the location returned by the vision system includes the rotational difference between the object recognized and the prototype taught. Thus, simply by moving the component from its present location to the location of the hole, the component will have the same orientation as the hole. This may seem obvious, but if there is a difference in the orientation of the prototypes, then the exact difference must

be known and compensated for before the component can be put into the hole.

2.5 Transformations

During the calibration of the camera, a transformation is developed. When the vision system sees an object, it calculates the location of the object's centroid relative to the origin of the camera. (The camera origin is the bottom left corner of the picture.) This relative location is given to the robot which must use the calibration transformation to convert it to a location relative to the robot's origin.

In the application of this research on the Adept™ robot, two other transformations were developed and found useful. The first is the centering transformation. This transformation places an object previously viewed, in the center of the camera's field of view and calculates the difference between the object centroid and a point near the center of the camera. Once this difference is found, the robot arm is moved this distance in the direction of the object.

The third transformation was developed as a result of the difficulties encountered with recognition of objects on the conveyors. (These difficulties are explained more fully later.) This transformation allows the robot to put down the part it is carrying and then move the arm so the part is in the center of the camera for identification purposes. This

transformation makes use of the calibration transformation (inverting it) and the centering transformation (using the relative camera center location). The importance of these transformations is discussed in the next chapter.

CHAPTER III

THE APPROACH

The issues dealt with in this chapter are mainly the details of the vision heuristic, database and state information representation, user interface, and the program logic for opportunistic scheduling.

3.1 The Picture Taking Heuristic

With all other vision parameters set, the maximum time allowance for vision system analysis can be determined. V.MAX.TIME is the variable that controls this important system parameter. The default value, as noted in Table 3, is 10 seconds, which almost insures an object will be recognized. In a manufacturing environment, however, where seconds are critical, 10 seconds is too long. To determine an optimal time, two issues were considered. The first was to determine experimentally how long the vision system actually required to identify each of the objects. After training the vision system with all the necessary prototypes, ten pictures of each of the objects were taken and analyzed. Each object was randomly located within the camera's field of view. Table 4 shows the resulting times.

These values are plotted in Figure 2. Notice that of the 200 pictures taken and analyzed, only 16 required more

than one second to process, and these were mainly the base assemblies where shadowing was a problem.

Table 4. Picture taking data

Trial	Circle	Section	Triangle	Square	Pentagon	Hexagon	Trapezoid	Cross	Oval	Star
1	0.26	0.29	0.37	0.24	0.29	0.75	0.34	0.46	0.42	0.50
2	0.24	0.50	0.35	0.32	0.27	0.54	0.38	0.46	0.91	0.46
3	0.21	0.59	0.38	0.98	0.29	0.94	0.80	0.64	1.28	0.59
4	0.26	0.35	0.50	0.30	0.30	0.61	0.53	0.40	0.34	0.45
5	0.26	0.59	0.26	0.27	0.32	0.51	0.62	0.35	0.40	1.12
6	0.26	0.42	0.32	0.29	0.29	0.58	0.75	0.46	1.58	0.43
7	0.24	0.30	0.46	0.24	0.40	0.69	0.45	0.43	0.62	1.25
8	0.22	0.38	0.38	0.30	0.30	0.64	0.61	0.53	0.74	0.85
9	0.22	0.50	0.43	0.27	0.26	0.75	0.43	0.48	0.43	0.43
10	0.22	0.34	0.34	0.27	0.27	0.64	0.38	0.56	0.40	0.74

Trial	Cir.base	Sec.base	Tri.base	Sq.base	Pen.base	Hex.base	Tra.base	Cro.base	O.base	St.base
1	0.29	0.51	0.50	0.82	0.50	0.43	0.50	0.50	0.77	0.66
2	0.22	0.51	0.62	0.80	0.42	0.40	0.53	0.48	0.50	0.80
3	0.29	1.28	0.37	0.88	0.42	0.35	0.50	0.50	0.51	1.50
4	0.32	0.74	0.51	1.39	0.85	0.29	0.75	0.59	0.45	0.90
5	0.32	0.62	0.35	1.01	0.64	0.40	0.51	0.77	0.78	2.30
6	0.29	0.38	0.88	0.72	0.51	0.30	2.02	0.50	0.51	0.80
7	0.27	0.38	0.46	0.85	0.38	0.74	0.43	0.59	0.35	0.91
8	0.26	0.40	1.26	1.34	0.50	0.58	0.58	0.51	1.28	0.59
9	0.27	0.38	0.82	0.96	0.53	0.30	0.37	0.54	0.54	0.56
10	0.64	0.40	4.05	0.93	0.46	0.42	0.40	0.48	1.73	1.18

The application parameter settings listed in Table 3 (V.MAX.TIME = 1.1 sec), produce a 0.92 probability of object recognition. Increasing V.MAX.TIME to 1.5 seconds would give a 0.97 probability of object recognition. The allowable time for analysis can be reduced and still maintain a relatively high probability of object recognition. As a result of these findings, the parameter V.MAX.TIME was set at 1.1 seconds.

The second issue considered arose as a result of

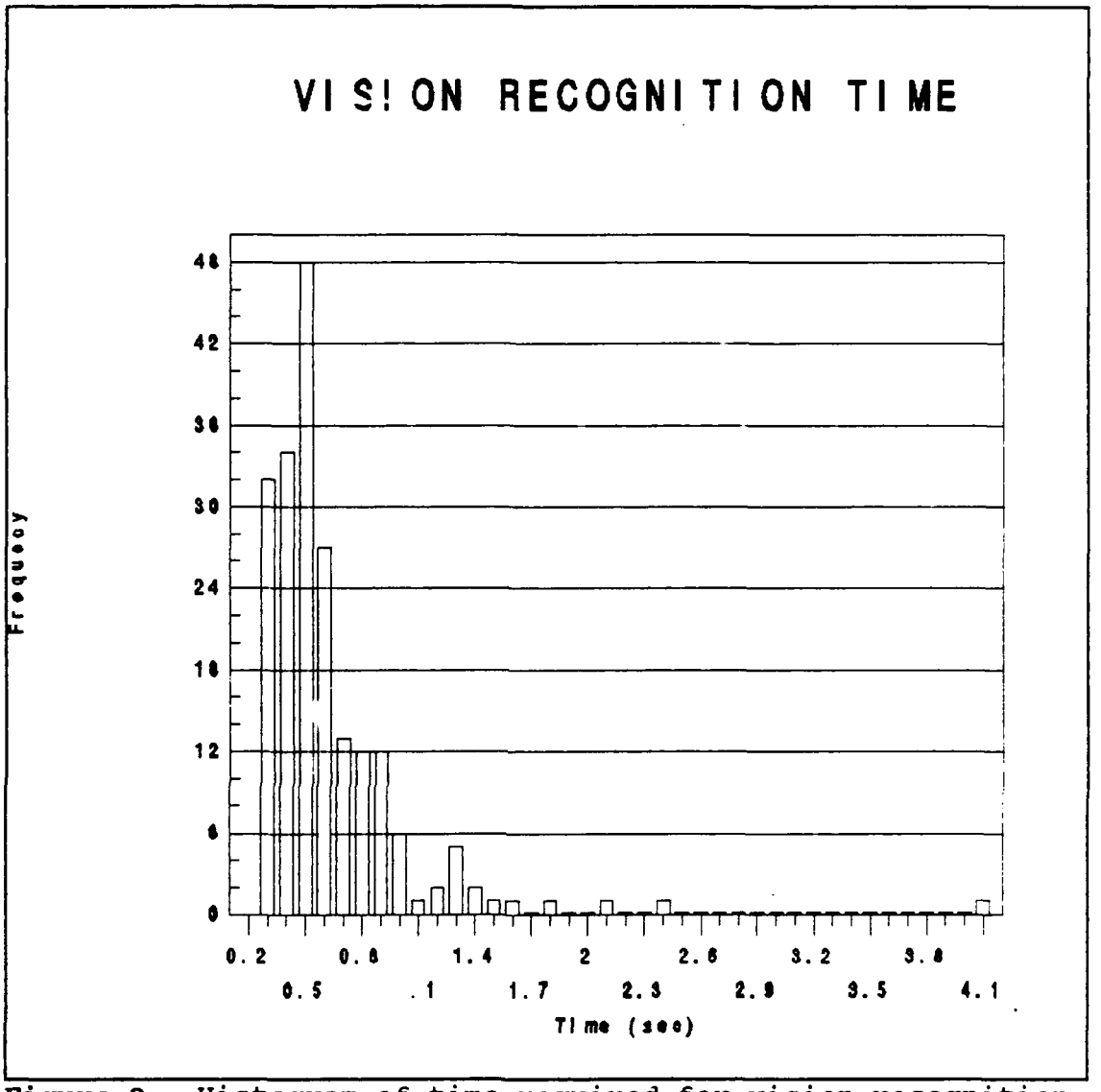


Figure 2. Histogram of time required for vision recognition.

restricting the time allowed for image processing, which causes the possibility of not recognizing a valid part. A closer look at Table 4 will show that even if a part required more than one second to be identified on the first trial, it could easily be recognized in much less than one second in the second trial. This leads to the vision system picture taking heuristic.

Figure 3 shows the logic diagram for the picture taking heuristic. This program is actually called as a subroutine by the main program. By first memorizing the initial location of the robot arm, the subroutine can move the arm and still return it to its original location when program control returns to the main program.

Taking an initial picture and storing the vision information about each of the objects viewed (e.g. object locations, name of prototype matched, and verification percentage) gives the program a starting point. If no object is viewed, the subroutine returns to the main program.

During training, the vision system was given a minimum recognition of 75 percent verification. This means the vision system will match a prototype with an image when only 75 percent of the image border matches the prototype border. This low percentage for matching (called verification) is very desirable, since parts can be randomly located within a region (the region being the camera field of view). However, when it comes to inserting components and performing quality control checks, the vision system must calculate the centroid and boundaries of the object very accurately.

The heuristic verifies that the image was matched with a prototype to at least the 90 percent level. If an object is seen, but does not meet this criteria, the object is centered in the camera view, and another picture is taken. If, on the second attempt, the image more closely matches the prototype, the stored vision information for this object is updated and

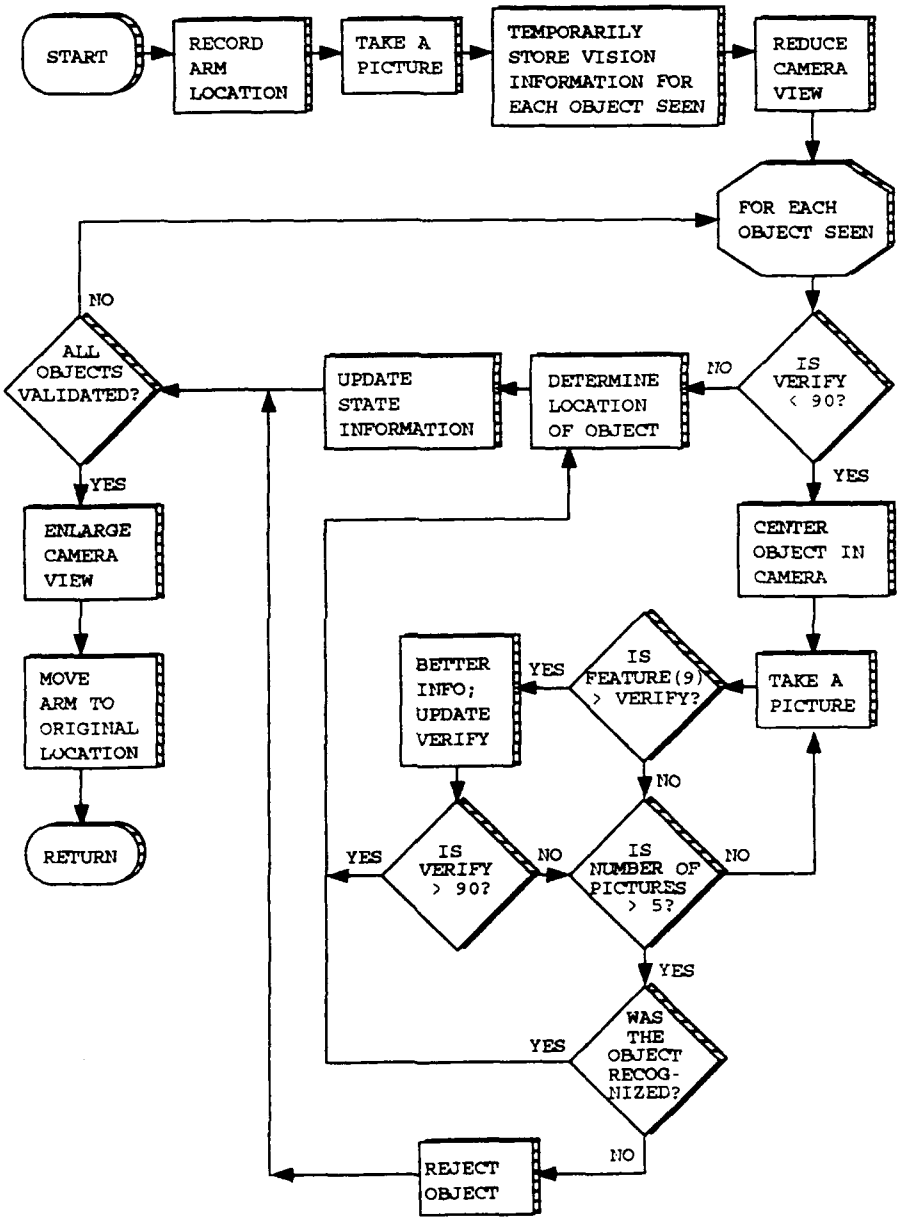


Figure 3. Picture taking heuristic.

the 90 percent test is made again. This loop is executed five times if the criteria are not met. Taking up to five pictures, each picture having a 0.92 probability of object recognition, gives an overall object recognition reliability of 0.9999967 for the vision system [23]. If, at the end of five attempts, the object is not recognized, it is determined to be "foreign" or outside of acceptable tolerances for the assembly. In this way the vision system conducts quality control checks while performing assembly operations. If an object is found to be out of tolerance, the robot picks it up and drops it into the reject region.

There is still a possibility of an object being recognized after centering and taking five pictures, but not to the 90 percent criteria. In these instances the vision information stored is used as if the object had passed the test.

Notice that after the initial picture, the camera view is reduced. This is accomplished by changing the parameters V.FIRST.COL, V.FIRST.LINE, V.LAST.COL, and V.LAST.LINE. The camera view is reduced just in case more than one object was seen in the initial picture, which, in turn, reduces vision analysis time. When an object is centered, it should be the only object fully visible to the camera. This eliminates the confusion that might result from seeing a different object the second time. Just before the subroutine returns to the main program, it enlarges the camera view back to full, and moves the robot arm back to its original location.

3.2 Multiple Assemblies

The robot is required to assemble four assemblies concurrently, taking advantage of the random delivery of components, decreasing the overall assembly time, and reducing the required buffer size. Assembling multiple products concurrently also introduces the need for priorities.

The priority of an assembly is based on two things. First, the database maintains a relative priority of each of the assemblies that the robot is capable of assembling. This information is based on which assemblies generally require a higher production rate. With a mix of assemblies arriving in a random sequence, there exists the possibility that any two of the assemblies will be processed concurrently. Priorities are used by the robot to resolve conflicts that arise when two or more base assemblies require the same component at the same time. Therefore, each assembly must have a unique priority value of one or greater. The lower the priority value the more important the assembly. A priority of zero means the robot will not process the assembly.

The priority value of any one base assembly can be decreased (made more important), depending on how long it has been in a work region awaiting completion. This time limit is based on how many other assemblies have been completed since the waiting base assembly arrived. The program maintains a running total of the number of products completed

since it was initially executed. A value of 10 completed assemblies is arbitrarily used as the limit for when the priority value of the waiting assembly is temporarily lowered. Priority information is part of the permanent database, and can easily be modified through the user friendly program menus (discussed in the next two sections).

3.3 The Database

The partial order representation of knowledge from artificial intelligence was used to represent the component precedence information in the robot. The two dimensional array PREC[X,Y] is this database. Consider the precedence diagram in Figure 4.

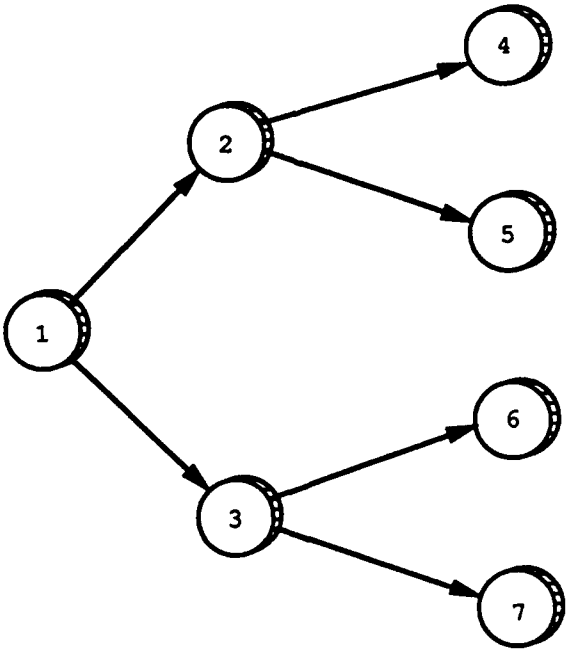


Figure 4. A precedence diagram of a simple assemble.

In list form, this precedence diagram would be represented as follows:

{(1),(7),(1,0),(2,1),(3,1),(4,2),(5,2),(6,3),(7,3)}

The first two elements describe the information (i.e. one assembly in the database, and the first assembly has seven constraints.) An individual constraint such as (4,2) indicates that component four requires component two be inserted first. This same knowledge would be represented in the array PREC[X,Y] as follows:

		Corresponding List Element
PREC[0,0] = 1	PREC[1,0] = 7	{(1),(7),
PREC[1,1] = 1	PREC[1,2] = 0	(1,0),
PREC[1,3] = 2	PREC[1,4] = 1	(2,1),
PREC[1,5] = 3	PREC[1,6] = 1	(3,1),
PREC[1,7] = 4	PREC[1,8] = 2	(4,2),
PREC[1,9] = 5	PREC[1,10] = 2	(5,2),
PREC[1,11] = 6	PREC[1,12] = 3	(6,3),
PREC[1,13] = 7	PREC[1,14] = 3	(7,3)}

Along with precedence information the database maintains the names of assemblies and components. The vision system returns the name of the prototype that matched the object viewed. Comparing this returned name to the names stored in the robot memory produced a method for retrieving all component and assembly information pertaining to the object located.

The precedence, part number, and name information constitute the permanent database. Each of these values can be easily modified through the database manager program. In this program the user can add, edit, or delete precedence

constraints. The database is capable of maintaining a large number of constraints (1 Megabyte storage). Depending on the number of components and complexity of assemblies, the robot could be programmed to assemble more than 100 different products.

3.4 Program Interface

The entire robotic program interface consists of menus. The top-level interface is represented by Figure 5 [24]. By executing program 'Schedule' the Main Menu will appear. The Main Menu has six options. Option number one is to execute program 'Assemble'. This program is the opportunistic scheduling program.

Main Menu option numbers two and three are database manager programs. Option number two controls assembly priority information, and option number three controls the component precedence information. To add, edit, or delete assemblies from the database, select option number three.

When the user is adding new information into the database, the robot assigns a new number to the new assembly (assembly number) and each of the new components (part number). This "part number" is used when setting precedence constraints.

Selecting option number four from the Main Menu displays the location menu. This menu allows the user to quickly move the robot arm-mounted camera to view any of the work regions.

Option number five of the Main Menu displays a menu of all the vision prototypes that are on the hard disk memory of the robot. A selected prototype will be loaded into active memory for immediate vision recognition capability.

3.5 Opportunistic Logic

The flow chart for the overall opportunistic logic can be seen in Figure 6. It should be noted that the program has no end but is an infinite loop. Thus, the robot is either working or waiting for delivery of parts.

3.5.1 State Information

The first thing the program does is to initialize the state information. The array REGION[X] maintains information about which base assembly is in each of the four working regions.

The component state information for each region is maintained by the array STATE[X,Y]. This array contains the precedence information for the assembly in REGION[X] from the array PREC[X,Y], but is updated each time a component is inserted (the dynamic precedence requirements). It should be noted that the X values are different in the STATE[X,Y] and PREC[X,Y] arrays. In PREC[X,Y], X is the assembly number, but in STATE[X,Y], X is the region number.

The arrays NEEDR[X] and NEEDC[X] are the arrays that

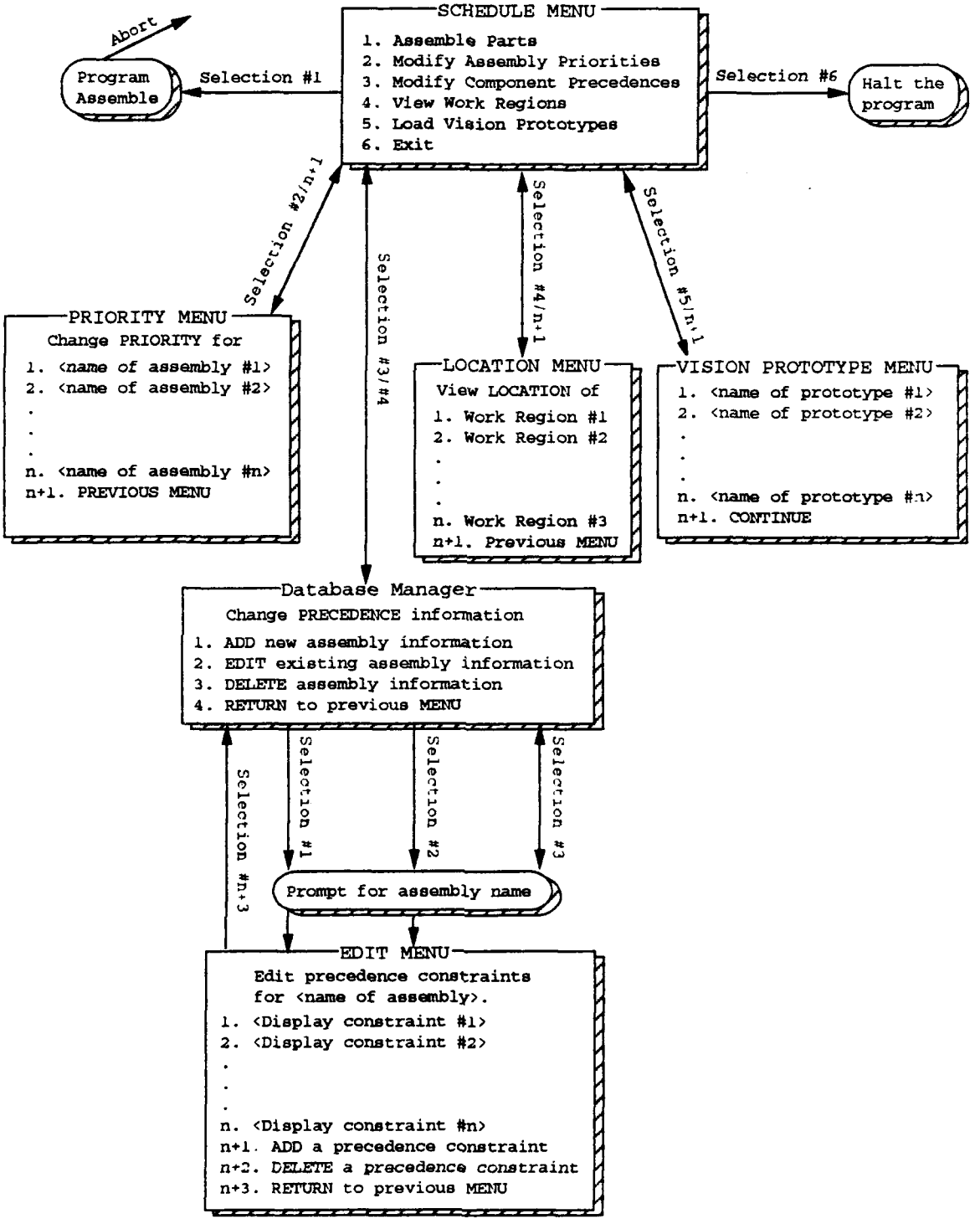


Figure 5. Robotic program interface.

maintain information about which regions have a requirement for which components. The variable NEEDC[0] is the current number of components required by all regions (the size of the arrays).

3.5.2 Programming Rules

Only six basic rules are used in this logic. More rules could be added later to further enhance program capability. For example, a rule for when a new base assembly is actually a partially completed assembly requiring rework/completion could be added. As seen in Figure 6 (diamond shaped figures represent rules), the first rule is the main rule. Rule one is, if a work region is empty, fill it with a new base assembly from the incoming conveyor. If a value in the array REGION[X] is zero, then there is no base assembly in that region. The robot attempts to keep all regions full in order to maintain the production rate at its highest.

Rule two is part of the procedure applied when rule one is executed (there is an empty work region). Rule two is, if a base assembly is in the pickup region, grasp it and move it to the empty work region. The conveyor that carries the base assemblies to the work area has an infrared sensor that signals to the robot when a new base assembly is present in the pickup region. The signal number is 10. Using the vision system, the robot identifies and locates the new base assembly in the pickup region. The robot then grasps the new

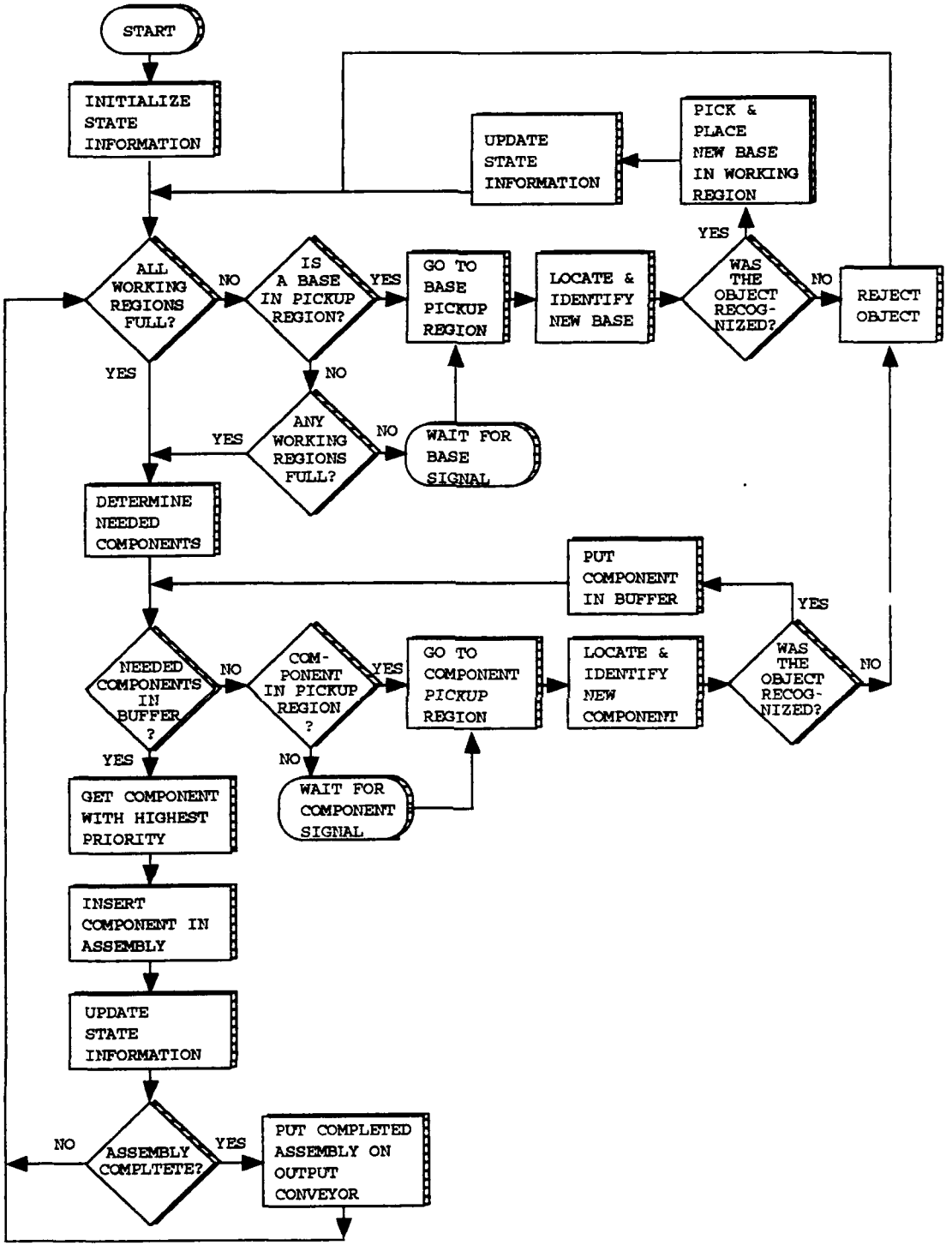


Figure 6. Flow chart of opportunistic logic.

assembly and places it in a vacant work region, and the program updates the STATE[X,Y] array for that region.

If there is no new assembly in the pickup region, the robot, referring back to the data for rule one, applies rule three. Rule three is, if a work region is full, determine needed components for the base assembly in that region. If there is work to be done, the robot goes to work. If not, the robot waits for the signal from the base assembly conveyor.

Rule four is, if needed components are in the buffer, insert them into the appropriate base assembly that has the lowest priority value. Array BUFFER[X] contains the component "part number" that is presently stored in buffer location X. This rule can be checked by examining the arrays NEEDC[X] and BUFFER[X] for matching component numbers.

If none of the needed components are in the buffer, rule five applies. Rule five states, if a component is in the pickup region, put it in a buffer location. The infrared sensor for the component conveyor provides signal number 11. If no component is in the pickup region, the robot will wait for the signal from the component conveyor. When a component is present, the robot arm moves to the pickup region and using the vision system locates and identifies the component. The component is placed in a buffer location and the array BUFFER[X] is updated by the program. After rule five, the program returns to rule four.

Rule six is the finished assembly rule, and says, if an

assembly is complete, place it on the finished product conveyor. This rule is checked by the STATE[X,Y] array. If this array contains no constraints for a given work region and the REGION[X] array indicates that an assembly is present in that work region, the assembly is considered complete (all precedence information has been satisfied). When an assembly is complete, it is placed on the finished product conveyor and the REGION[X] array is updated to represent the empty work region.

After applying rule six, the program returns to rule one. These rules with their respective procedures maintain production but allow the production rate to change dynamically.

CHAPTER IV

RESULTS AND CONCLUSIONS

The results of this research were very positive. In this chapter, successes and short comings are discussed.

4.1 Discussion of Advantages

In many applications, one robot is dedicated to the assembly of one product. Sometimes, the robot may have programs stored in memory for other assembly operations, but, even then, it is restricted to assembling one product at a time. In order for the robot to assemble a different product, it must first load the respective program. Often, fixtures must also be changed, requiring lengthy set up times.

To fully understand the possible advantages of opportunistic assembly, reconsider the following example. A particular manufacturer has five separate product lines. Each of these lines has a robotic workstation dedicated to final assembly. Each robot has been programmed to assemble its product in a fixed, but optimal way. Components for assembly are delivered to prescribed locations by specialized part feeders, and base assemblies are loaded manually by a technician.

If one of the robots breaks down or is due for regular

maintenance, assembly of the product assembled by that robot would come to a complete stop until repairs were finished. If a placement error occurred at one of the workstations, the line would stop until the error was corrected by the human technician. If a component was fed to the robot out of sequence, assembly of that product would completely stop until the error was corrected.

If, however, each of the five robots were given vision capability and programmed to use opportunistic scheduling, the situation would change. There would be no placement errors, except those due to vision inaccuracy, because all parts being delivered can be randomly located and randomly oriented in the pickup regions. Specialized part feeders would not be necessary because simple forms of part feeding (i.e. conveyors) can now be utilized.

Production stoppage due to out of sequence components, would never occur, because the components and base assemblies can be in any sequence. Furthermore, if each of the robots were programmed to use a database containing the assembly knowledge of each product, the five distinct workstations would become five mixed workstations, each capable of assembling any of the products being assembled. With mixed workstations, if one robot stopped for maintenance, the other four robots would still be producing a mixture of all the products. Thus, the production of any one product would not stop due to one robot stopping. There is also the savings that result from the need for fewer fixtures, and the time

savings due to the elimination of pre-processing of parts.

4.2 Discussion of Short Comings

Even though the above advantages are significant, there are a few short comings to be discussed. Short comings with the design of the work cell and the equipment used will be discussed first, and is followed by a discussion of difficulties with opportunistic scheduling.

4.2.1 Short Comings of the Apparatus

Probably the greatest difficulty encountered was the recognition of components on the conveyor. If the vision prototypes are taught from the conveyor, then the vision system can recognize the parts on the conveyor but not in the work area (The work area and conveyor are at different heights in the test case studied). The converse is true also, if the prototypes are taught from the work area, the vision system can not recognize them on the conveyor. (If the conveyor and work area had been designed to be the same height, this inconvenience is eliminated.)

There is a significant savings in assembly time if the vision system recognized the parts as they arrive on the conveyor. However, for some unknown reason, when an object was on the conveyor, the vision system always miscalculated

the centroid, causing the robot to pick up the object incorrectly. This error is about 1/2 of an inch, so the robot can still pick up the object (due to the size of the objects being used).

To temporarily overcome this difficulty, the vision system is used only to locate an object on the conveyor. Once located, the robot places the object in a buffer location, and uses the "step back" transformation to move back and take another picture for identification purposes. It is suspected that there is a problem with lighting on the conveyor. (The light from the infrared sensor is visible to the camera and may cause uneven lighting.)

Many unwanted shadows caused problems with proper placement of components. A part not centered in the picture, would have shadows causing incorrect calculation of the centroid. The miscalculation was sometimes off by as much as 1/4 on an inch which was too large to correctly insert the component. The centering transformation was used to reduce this error. The location error could be almost eliminated by changing the lighting methods and the relative position of the light sources. More light in the correct direction would minimize shadowing and produce a more evenly lighted work area.

4.2.2 Short Comings of the Approach

There are a few difficulties with opportunistic

scheduling. For example, the definition of an opportunity and what creates an opportunity must be known and programmed into the robot in advance. In this research, opportunities were restricted in the robot workstation to the random arrival of base assemblies and components. However, if the technician picked up a component that had fallen off the conveyor and put it in the buffer, an error would occur because this "opportunity" was not allowed by the present production rules.

A method must be devised to evaluate each of the opportunities. Assembly priorities, component precedence information, the first-come-first-serve inventory discipline, and the production rules for assembly were used in this research to evaluate opportunities. Making provisions for more types of opportunities causes the recognition and evaluation of an opportunity to become more complex.

Knowing in advance which alternatives will be available due to an opportunity is a difficulty people face every day. Opportunistic scheduling is no different. The programmer must know what kind of opportunities would come to a robot in the future. He would have to program a method for evaluating each of them and know which alternatives will be available. All this must be given to the robot before it can take advantage of an opportunity.

The method used to represent knowledge can also create difficulties. In this research the assembly precedence information was maintained as arrays in the form of a partial

order. Methods available for this implementation were limited by VALII, the language of the robot. However, there are several, including better, alternative methods for knowledge representation. One example would be an AND/OR graph representation that would reduce the size of the database needed to represent the precedence information [25].

4.3 CONCLUSIONS

The goal of this research was to combine computerized vision and artificial intelligence programming in an application of robotic assembly that would use opportunistic scheduling. An Adept™ single manipulator robot was provided with the capability of visually recognizing random opportunistic events. The objective of a heuristic for taking pictures was accomplished, and the heuristic significantly improve overall reliability of object recognition for the vision system. The requirements for specialized part feeders and pre-processing of parts were virtually eliminated, allowing all base assemblies and components to be delivered randomly on conveyor belts to the robot work cell.

The robot was given basic assembly knowledge using the production rule methodology from artificial intelligence and assembly precedence information using a database of partial order sets. Dynamic state information was also maintained by the program. The objective to use artificial intelligence

programming was thereby fulfilled. The robot was also given the capability of assembling a mix of products and assembling multiple products concurrently. The robot was thus able to assemble products in any feasible way and schedule an optimal assembly plan according to the random arrival of parts.

There was an objective to develop a user friendly interface with the robot for flexible production of multiple assemblies. This objective was accomplished with the database and the Database Manager program menus.

The last objective was to implement the above interface, vision heuristic, database, and rule-based program on the Adept single arm robot with arm-mounted camera. All these things were implemented and the results were very favorable.

The following is a discussion of the three positive contributions of this research.

1. *The need for fixtures and restrictive part feeders was drastically reduced.* Using the arm-mounted vision system, part location and orientation do not have to be pre-processed to conform to the robots programming. This, in turn, reduces the time required for product assembly. Also, the manual placement of base assemblies is eliminated.

2. *Vision recognition time was reduced using the heuristic developed.* The heuristic reduced the time required for object recognition and improved the overall object recognition reliability of the vision system. Quality control can be accomplished more consistently during the assembly process (with no added processing time). Errors due

to improper placement of parts were also nearly eliminated.

3. *One robot can assemble multiple products concurrently.* It is no longer necessary to dedicate a robot to one product line. Using a database of assembly precedence information and multiple working regions, the set up times between diversified products are almost non-existent. Programming in the production rules for opportunistic scheduling permits production scheduling and sequencing to dynamically change without any necessary changes to the robot cell.

Thus, it can be seen that combining computer vision and artificial intelligence in the implementation of opportunistic scheduling for robotic assembly produces a very powerful and flexible tool for modern assembly.

There are still a great many issues to be studied that would improve this application to the point of possible implementation in a real assembly plant. Further research could include:

1. Incorporate opportunistic scheduling on a multiple arm robot.

2. Simulate opportunistic assembly to determine the extent of advantages over other methods, and the effect of randomly sequenced components.

3. Study the effects of assembling multiple products concurrently, and the effect of the number of work regions.

4. Develop an application of opportunistic scheduling for assembly rework and the effects of partially assembled products being fed to the robot at the same time as new base assemblies.

REFERENCES

[1] Hoard P. Emerson and Douglas C. E. Naehring, *Origins of Industrial Engineering*. Atlanta, Ga: Institute of Industrial Engineers, 1988.

[2] Mikell P. Groover, *Automation, Production, Systems, and Computer-Integrated Manufacturing*. Englewood Cliffs, NJ: Prentice-Hall, Inc., 1987.

[3] Isaac Asimov and Karen A. Frenkel, *Robots*. New York, NY: Nightfall, Inc. and Karen A. Frenkel, 1985.

[4] Mikell P. Groover, Mitchell Weiss, Roger N. Nagel, and Nicholas G. Odrey, *Industrial Robotics*. McGraw Hill Publishing Company, 1986.

[5] Ulrich Rembold, Christian Blume, and Ruediger Dillman, *Computer-Integrated Manufacturing Technology and Systems*. New York, NY: Marcel Dekker, Inc., 1985.

[6] A. L. Giacobbe, "Diskette Labeling and Packaging Systems Features Sophisticated Robot Handling", Robotics Today, pp. 73-75, April 1984.

[7] D. Chapman, "Planning for Conjective Goals", Artificial Intelligence, Vol 32, No. 3, pp. 333-377, July 1987.

[8] T. L. De Fazio and D. E. Whitney, "Simplified Generation of All Mechanical Assembly Sequences", IEEE Journal of Robotics & Automation, pp. 640-658, Dec 1987.

[9] J. D. Wolter, "On the Automatic Generation of Assembly Plans", Proc. IEEE Int. Conf. Robotics and Automation, pp. 62-68, 1989

[10] Kai-I Huang, Cesar O. Malave, and Milden J. Fox, Jr., "A Review of Assembly Sequence Planning Problems", Department Industrial Engineering, Texas A&M University.

[11] Yasuhiro Monden, *Toyota Production System*. Norcross, Ga: Industrial Engineering and Management Press, 1983.

[12] David D. Bedworth and James E. Bailey, *Integrated Production Control Systems*. New York, NY: John Wiley & Sons, 1987.

[13] B. R. Fox, K. G. Kempf, "Opportunistic Scheduling for robotic Assembly", Proc. IEEE Int. Conf. Robotics & Automation, 1985, pp. 880-889.

[14] Barbara Hayes-Roth, et. al., "Human Planning Processes", A Report prepared for the Office of Naval Research, December 1980.

[15] Steven L. Tanimoto, *The Elements of Artificial Intelligence.* Rockville, MD: Computer Science Press Inc., 1987.

[16] Barbara Hayes-Roth, Frederick Hayes-Roth, "Cognitive Processes in Planning", A Report prepared for Office of Naval Research, December 1978.

[17] T. O. Prenting, and R. M. Battaglin, "The Precedence Diagram: A Tool for Analysis in Assembly Line Balancing", Journal of Industrial Engineering, Vol. 15, No. 4, pp. 208-211, 1964.

[18] Lynwood A. Jonson and Douglas C. Montgomery, *Operations Research in Production Planning, Scheduling, and Inventory Control.* New York, NY: John Wiley & Sons, 1974.

[19] A. P. Ambler, H. G. Barrow, C. M. Brown, R. M. Burstall, and R. J. Popplestone, "A Versatile System for Computer-Controlled Assembly", Artificial Intelligence, Vol. 6, pp. 129-156, 1975.

[20] Interim Report #3, *Research for Intelligent* Task Automation, Air Force Contract #F33615-82-C-5092, July 15, 1983.

[21] *AdeptVision Users Manual*, Adept Technology Inc., 1986.

[22] *Vision Training Course Manual*, Adept Technology Inc., 1987.

[23] K. C. Kapur and L. R. Lamberson, *Reliability in Engineering Design.* New York, NY: John Wiley & Sons, 1977.

[24] David L. Parnas, "On the Use of Transition Diagrams in the Design of a User Interface For an Interactive Computer System", Proc. of the 24th National Conf. of the ACM, pp. 379-385, 1969.

[25] Luiz S. Homem De Mello and Arthur C. Sanderson, "And/Or Graph Representation of Assembly Plans", IEEE Trans. on Robotics & Automation, pp. 188-199, Apr 1990.

APPENDIX A

LISTING OF OPPORTUNISTIC PROGRAM

A.1 Program Schedule

```
.PROGRAM schedule()
 10   TYPE $clear.screen,/C4
      TYPE /X19,"*********** SCHEDULING MENU ***********"
      TYPE /X19,"*                                      *"
      TYPE /X19,"*    1. Assemble Parts                 *"
      TYPE /X19,"*    2. Modify Assembly Priorities     *"
      TYPE /X19,"*    3. Modify Component Precedences   *"
      TYPE /X19,"*    4. View Work Regions              *"
      TYPE /X19,"*    5. Load Vision Prototypes         *"
      TYPE /X19,"*    6. Exit                           *"
      TYPE /X19,"*                                      *"
      TYPE /X19,"****************************************"
      TYPE " "
      PROMPT "                              ENTER CHOICE: ",menu
      CASE menu OF
        VALUE 1:
          CALL assemble()
        VALUE 2,3:
          CALL database()
        VALUE 4:
          CALL position()
        VALUE 5:
          CALL start.up()
        VALUE 6:
          HALT
        ANY
          TYPE /C1,/X10,"Please enter an integer between 1 and
               5 inclusive.",/C1
          PROMPT "          Press RETURN to continue. ",menu
      END
      GOTO 10
.END
```

A.2 Program Assemble

```
.PROGRAM assemble()
; initialize state information (all 0's)
```

```
; region[x]     - assembly number for the assembly in region x
; part[x,y]     - part number of component y in assembly x
; produced[x]   - arrival sequence number for region[x]
; produced[0]   - cumulative total of products assembled
; state[x,y]    - dynamic precedence information for region[x]
; prec[x,y]     - permanent precedence info - all assemblies
; prior[x]      - dynamic priorities for region[x]
; pri[x]        - permanent priorities - all assemblies
; needc[x]      - part number for needed component
; needc[0]      - total number of needed components
; needr[x]      - work region needing corresponding needc[x]
; buffer[x]     - part number - component in buffer location x
; comp_loc[x]   - location of component x (null if not known)
; insert_loc[x]- location of hole x (null if not known)
; base_loc[x]   - location of base assembly
; pic_loc[x]    - location for camera to view work regions
; num_regions   - total number of work regions (4)
; comp_con      - conveyor location of component pickup region
; finish_con    - conveyor location for product dropoff region
; base_con      - conveyor location for assembly pickup region
; cum_num_parts- cumulative number of components in database
; $assembly[x] - name of assembly x in the database
; $comp[x,y]    - name of components y in assembly x

 1    LEFTY
      MOVE home
      BREAK
      x = 1
      WHILE (x <= num_regions) DO
        region[x] = [x]  ;region[x] = 0 for implementation
        produced[x] = 0
        buffer[x] = 0
        needc[x] = 0
        needr[x] = 0
        state[x,0] = 0
        SET base_loc[x] = NULL
        y = 1
        WHILE (y <= part[x,0]) DO
          state[x,y*2-1] = 0
          state[x,y*2] = 0
          y = y+1
        END
        x = x+1
      END
      x = 1
      WHILE (x <= 10) DO
        SET comp_loc[x] = NULL
        SET insert_loc[x] = NULL
        x = x+1
      END
      produced[0] = 0

; Load the STATE arrays with the experimental assembly info
```

```
      x = 1
      WHILE (x <= num_regions) DO
        y = 0
        WHILE (y <= prec[region[x],0]*2) DO
          state[x,y] = prec[region[x],y]
          y = y+1
        END
        x = x+1
      END

; Locate the base assemblies placed in the work regions.

      x = 1
      WHILE (x <= num_regions) DO
        prior[x] = pri[region[x]]
        IF prior[x] > 0) THEN
          MOVE pic_loc[x]
          BREAK
          flag = 10
          CALL sh.look()
        END
        x = x+1
      end
      flag = 0

; Now back to the program - How many regions are full?

 5    x = 1
      full = 0
      max_prior = 0
      WHILE (x <= num_regions) DO
        IF (region[x] > 0) THEN
          full = full+1
          IF (prior[x] > max_prior) THEN
            max_prior = prior[x]
          END
        END
        IF (produced[0] >= produced[x]+10) THEN ; bump prior
          prior[x] = prior[x]-1
          produced[x] = produced[0]
          IF (prior[x] <= 0) AND (pri[region[x]] > 0) THEN
            prior[x] = 1
          END
        END
        x = x+1
      END

; Are all the work regions full?

      IF (full == num_regions) THEN

; Determine needed components & rank them by region priority
```

```
10      p = 1
        z = 1
        needc[0] = 0
        WHILE (p <= max_prior) DO
          y = 1
          WHILE (y <= num_regions) DO
            IF (prior[y] == p) THEN
              x = 1
              WHILE (x <= state[region[y],0]) DO
                IF (state[region[y],x*2] == 0) THEN
                  needc[0] = needc[0]+1
                  needc[z] = state[region[y],x*2-1]
                  needr[z] = y
                  z = z+1
                END
                x = x+1
              END
            END
            y = y+1
          END
          p = p+1
        END

; Are the needed components in the buffer?

15      x = 1
        WHILE (x <= needc[0]) DO
          y = 1
          WHILE (y <= 10) DO  ;buffer size is ten
            IF (buffer[y] == needc[x]) THEN
              CALL pick_place(comp_loc[needc[x]],
                              insert_loc[needc[x]])

; Now update state information

              k = 1
              WHILE (k <= state[needr[x],0) DO

; Erase the constraints that were just fulfilled

                IF (state[needr[x],k*2-1] == needc[x]) AND
                    state[needr[x],k*2] == 0) THEN
                  l = k*2-1
                  WHILE (l <= state[needr[x],0]*2) DO
                    IF (l+2 > state[needr[x],0]*2) THEN
                      state[needr[x],l] = 0
                    ELSE
                      state[needr[x],l] = state[needr[x],l+2]
                    END
                    l = l+1
                  END
                  state[needr[x],0] = state[needr[x],0]-1
```

```
; Update constraints requiring the component just inserted

                m = 1
                WHILE (m <= state[needr[x],0]) DO
                  IF (state[needr[x],m*2] = needc[x]) THEN
                    state[needr[x],m*2] = 0
                  END
                  m = m+1
                END
              END
              k = k+1
            END
            SET comp_loc[needc[x]] = NULL
            SET insert_loc[needc[x]] = NULL
            buffer[y] = 0
            GOTO 20  ;Jump out of loops
          END
          y = y+1
        END
        x = x+1
      END

; Is a component in the pick up region?

      IF SIG(11) THEN
        LEFTY
        MOVE comp_con
        BREAK
30      CALL take.a.picture()
        VLOCATE (0) $part, vloc
        IF VFEATURE(1) THEN
          SET temp1 = picloc:RZ(-jt[4]):vtran[1]:vloc
          SET objloc = shift(temp1 BY 0,0,-69)

; Put the new component in an empty buffer location

          x = 1
          WHILE (x <= 11) DO  ;buffer size is 10
            IF (x == 11) THEN
              y = 1
              WHILE (y < x) DO

                TYPE $CHR(7)
                buffer[y] = 0
                y = y+1
              END
              TYPE /C1,/X5,"*** BUFFER OVERLOAD ***"
              PROMPT "      EMPTY BUFFER AND PRESS RETURN"
              GOTO 1
            END
            IF (buffer[x] == 0) THEN
              APPRO objloc, 40
              SPEED 20
```

```
                MOVES objloc
                CLOSEI
                DEPART 40
                APPRO buffer_loc[x], 160
                SPEED 20
                MOVE buffer_loc[x]
                OPENI
                DEPART 186.95
                BREAK
                GOTO 35  ;Jump out of loop
              END
              x = x+1
            END

; Now identify the component just put in the buffer.

 35         HERE #temp1
            HERE temp1
            DECOMPOSE a[1] = #temp1
            MOVE temp1:RZ(-a[4]):ivtran[1]:icent
            BREAK
            PARAMETER V.FIRST.LINE = 100
            PARAMETER V.LAST.LINE = 383
            PARAMETER V.FIRST.COL = 150
            CALL sh.look()
            PARAMETER V.FIRST.LINE = 1
            PARAMETER V.LAST.LINE = 483
            PARAMETER V.FIRST.COL = 1

; Is the new component needed?

              buffer[x] = seen
              GOTO 15
            ELSE
              GOTO 30
            END
          ELSE  ;Wait for a component to come.
            WAIT 11
            GOTO 25
          END

; Is an assembly complete?

 20     x = 1
        WHILE (x <= num_regions) DO
          IF (state[x,0] == 0) AND (region[x] <> 0) THEN
            CALL pick_place(base_loc[region[x]], finish_con)
            region[x] = 0  ;reset region value to zero
            GOTO 5
          END
          x = x+1
        END
        GOTO 5
```

```
; If all regions are not full, get another base assembly.

     ELSE
 40    IF SIG(10) THEN
         MOVE base_con
         BREAK
         CALL sh.look
         x = 1
         WHILE (x <= num_regions) DO
           IF (region[x] == 0) THEN
             IF (DX(base_loc[seen] <> 0) THEN
               CALL pick_place(base_loc[seen], region_loc[x])
               region[x] = seen
               prior[x] = pri[seen]
               SET base_loc[seen] = region_loc[x]
               x = 5  ;Jump out of loop
             END
           END
           x = x+1
         END

;If no base is in the pickup region, is a work region full?

       ELSE
         IF (full > 0) THEN
           GOTO 10

; If all work regions are empty wait for a base assembly.

         ELSE
           WAIT 10
           GOTO 40
         END
       END
     END
.END
```

A.3 Program Pick_place

```
.PROGRAM pick_place(from, to)
     APPRO from, 50
     SPEED 20
     MOVES from
     CLOSEI
     DEPART 50
     APPRO to, 50
     SPEED 20
     MOVES to
     OPENI
```

```
     DEPART 50
     BREAK
.END
```

A.4 Program Database

```
.PROGRAM database()
 20  TYPE $clear.screen
     IF (menu == 3) THEN
       TYPE /C4,/X15,"************** DATABASE MANAGER
**************"
       TYPE /X15,"*",/X44."*"
       TYPE /X15,"*",/X8,"Change PRECEDENCE information",
/X7, "*"
       TYPE /X15,"*",/X44."*"
       TYPE /X15,"*   1. ADD new assembly information", /X10,
"*"
       TYPE /X15,"*   2. EDIT existing assembly information",
/X5,"*"
       TYPE /X15,"*   3. DELETE assembly information", /X11,
"*"
       TYPE /X15,"*   4. RETURN to previous MENU", /X15, "*"
       TYPE /X15,"*",/X44."*"
       TYPE /X15,"********************************
**************",/C1
       PROMPT "                          ENTER CHOICE: ",data
       TYPE $clear.screen
       CASE data OF

; The following is for adding a new assembly to the database.

         VALUE 1:
           IF (DEFINED(part[0,0])) THEN
             part[0,0] = part[0,0]+1
           ELSE
             part[0,0] = 1
           END
           prec[0,0] = part[0,0]
           num = part[0,0]
           TYPE /C2
           PROMPT "          Enter a name for the new
assembly: ", $assembly[num]
           TYPE " "
           PROMPT "          Enter the number of components
in the new assembly: ",part[num,0]
           TYPE " "
           prec[num,0] = part[num,0]

; Give each new component a part number.
```

```
            x = 1
            WHILE (x <= part[num,0]) DO
              cum_num_parts = cum_num_parts+1
              prec[num,x*2-1] = cum_num_parts
              prec[num,x*2] = 0
              part[num,x] = cum_num_parts
              x = x+1
            END

; Ask for names (to match vision prototypes) of components.

            x = 1
            WHILE (x <= part[num,0]) DO
              TYPE /X10,"Enter a name for component number",
x, /S
              PROMPT ": ",$comp[num,x]
              x = x+1
            END
            flag = 0
            GOTO 30  ;Goto 'EDIT MENU'

;The following is the EDIT MENU - editing precedence info.

          VALUE 2:
            flag = 0
 40         TYPE /C2,/X10,"Enter the name of the assembly to
EDIT",/S
            PROMPT ": ",$ans
            num = 1
            WHILE (num <= part[0,0]) DO
              IF $assembly[num] == $ans GOTO 30
              num = num+1
            END
            TYPE /C1,/X10,"Sorry, assembly ",$ans," was not
found in the database.",/C1
            PROMPT "          Press RETURN to continue. ",$ans
            GOTO 20  ;Return to 'DATABASE MANAGER'
 30         TYPE $clear.screen
            IF (flag == 0) THEN
              TYPE /C2,/X20,"*************** EDIT MENU
***************",/C1
              TYPE /X23,"   Edit precedence constraints  "
            ELSE
              TYPE /C2,/X20,"************** DELETE MENU
**************",/C1
              TYPE /X23,"   Delete precedence constraints  "
            END
            TYPE /X23,"   for ",$assembly[num],/C1
            x = 1
            WHILE (x <= prec[num,0]) DO
              IF (prec[num*2] > 0) THEN
                TYPE /X22,x,". Component #",prec[num,x*2-1], "
requires component #", prec[num,x*2]
```

```
              ELSE
                TYPE /X22,x,". Component #",prec[num,x*2-1], "
requires no components"
              END
              IF (x == prec[num,0]) THEN
                IF (flag == 0) THEN
                  TYPE /X22,x+1,". ADD a precedence
constraint"
                  TYPE /X22,x+2,". DELETE a precedence
constraint"
                  TYPE /X22,x+3,". RETURN to previous MENU"
                ELSE
                  TYPE /X22,x+1,". RETURN to previous MENU"
                END
              END
              x = x+1
            END
            TYPE /X35, "Enter Choice",/S
            PROMPT ": ",edit
            IF (flag == 1) THEN
              IF (edit == prec[num,0]+1) THEN
                flag = 0
                GOTO 30  ;Goto 'EDIT MENU'
              ELSE
                GOTO 60  ;Goto 'DELETE a constraint'
              END
            END
            CASE edit OF

; Add a basic constraint then GOTO the edit menu to edit.

              VALUE prec[num,0]+1:
                flag = 2
                prec[num,0] = prec[num,0]+1
                TYPE /C1,/X10,"Enter the name of the component
constrained",/S
                PROMPT ": ",$ans
                x = 1
                WHILE (x <= prec[num,0]) DO
                  IF $comp[num,x] == $ans THEN
                    prec[num,prec[num,0]*2-1] = x
                    edit = x
                    GOTO 50  ;Goto 'Edit a constraint'
                  END
                  x = x+1
                END
                edit = prec[num,0]
                $comp[num,edit] = $ans
                prec[num,edit*2-1] = prec[num,0]
                GOTO 50  ;Goto 'Edit a constraint'

; Delete a constraint & update constraints dependent on it.
```

```
            VALUE prec[num,0]+2:  ;DELETE a constraint
60            IF (flag == 0) THEN
                flag = 1
                GOTO 30  ;Goto 'DELETE MENU'
              ELSE
                prec[num,edit*2] = 0
                x = 1
                WHILE (x <= prec[num,0]*2) DO
                  IF (prec[num,edit*2-1] == prec[num,x]) AND
                     (x <> edit*2-1) THEN
                    y = edit*2-1
                    WHILE (y <= prec[num,0]*2) DO
                      IF (y+2 > prec[num,0]*2) THEN
                        prec[num,y] = 0
                      ELSE
                        prec[num,y] = prec[num,y+2]
                      END
                      y = y+1
                    END
                    prec[num,0] = prec[num,0]-1
                    GOTO 30  ;Goto 'DELETE MENU'
                  END
                  x = x+2
                END
                GOTO 30  ;Goto 'DELETE MENU'
              END

; Return to the 'DATABASE MANAGER'.

            VALUE prec[num,0]+3:
              GOTO 20
          END

; The following is edit a constraint.

 50       IF (edit <= prec[num,0]) THEN
            TYPE /C1,/X5,"Enter the component number
required by component #",part[num,edit],/S
            PROMPT ": ",ans

; Is the component required in the assembly?

            x = 1
            WHILE (x <= part[num,0]) DO
              IF (part[num,x] == ans) THEN
                IF (flag == 0) THEN
                  prec[num,edit*2] = part[num,x]
                ELSE
                  prec[num,prec[num,0]*2] = part[num,x]
                END
                flag = 0
                GOTO 30  ;Return to EDIT MENU
              END
```

```
              x = x+1
            END
            TYPE /C1,/X10,"Sorry, component #", ans, " is
not used in ", $assembly[num], /C1
            PROMPT "          Press RETURN to continue. ",
$ans
            GOTO 30  ;Return to EDIT MENU
          END

; The following deletes an entire assembly from the database.

        VALUE 3:
 70       TYPE $clear.screen
          TYPE /C4,/X20,"*************** DELETE MENU
***************"
          TYPE /C1,/X25,"DELETE an entire assembly from"
          TYPE /X25,"the database.",/C1
          x = 1
          WHILE (x <= prec[0,0]) DO
            TYPE /X23,x,". ",$assembly[x]
            x = x+1
          END
          TYPE /X23, x,"RETURN to previous MENU"
          TYPE /C1,/X20,
"****************************************"
          TYPE /C1,/X35,"ENTER CHOICE",/S
          PROMPT ": ", delete
          IF (delete > prec[0,0]) GOTO 20  ;Return
          x = delete
          WHILE (x < prec[0,0]) DO
            $assembly[x] = $assembly[x+1]
            prec[x,0] = prec[x+1,0]
            y = 1
            WHILE (y <= prec[x,0]*2) DO
              prec[x,y] = prec[x+1,y]
              IF (y <= prec[x,0]) THEN
                $comp[x,y] = $comp[x+1,y]
              END
              y = y+1
            END
            x = x+1
          END
          prec[0,0] = prec[0,0]-1
          GOTO 70  ;Return to DELETE MENU.

; Return to the 'SCHEDULING MENU'.

        VALUE 4:
          GOTO 100  ;Goto END of this program
      END

; The following menu allows changing of assembly priorities.
```

```
     ELSE
       TYPE /X4,"NOTE: For priorities, the lower the number,
the more important"
       TYPE /X10,"the assembly.  No two assemblies should
have the same priority."
       TYPE /X10,"A priority of 0 means the assembly will NOT
be assembled."
       TYPE /C1,/X10,"******************** PRIORITY MENU
********************",/C1
       TYPE /X30,"Change PRIORITY for",/C1
       x = 1
       WHILE (x <= part[0,0]) DO
         IF (x == part[0,0]) THEN
           TYPE /X14,x,". ",$assembly[x],/X2,pri[x],/X6,x+1,
". Previous MENU"
         ELSE
           IF (part[0,0] > 0) THEN
             TYPE /X14,x,". ",$assembly[x],/X2,pri[x],/X6,
x+1,". ",$assembly[x+1],/X2,pri[x+1]
             IF (x+2 > part[0,0]) THEN
               TYPE /X14,x+2,". Previous MENU"
             END
           ELSE
             TYPE /X14,x,". Previous MENU"
           END
         END
         x = x+2
       END
       TYPE /C1,/X10,"************************************
********************",/C1
       TYPE /X32,"ENTER CHOICE",/S
       PROMPT ": ", assembly
       IF (assembly > part[0,0]) GOTO 100  ;Return
       TYPE /C1,/X10,"Enter the new PRIORITY for the
assembly",/S
       PROMPT ": ", pri[assembly]
       GOTO 20  ;Return to 'PRIORITY MENU'
 100 END
.END
```

A.5 Program Position

```
.PROGRAM position()
     num = 0
 20  TYPE $clear.screen
     IF (num == 1) THEN
       TYPE /C1,/X19,"The camera is viewing work region
number", loc
     ELSE
```

```
      TYPE /C1,/X10,"The following menu allows you to view
the different work"
      TYPE /X10,"regions.  By choosing a number in the menu
below, the robot will"
      TYPE /X10,"position the camera over the region
specified.  The assembly can"
      TYPE /X10,"be located anywhere in the camera's field
of view."
    END
    TYPE /C2,/X25,"******* LOCATION MENU *******"
    TYPE /X25,"*                              *"
    TYPE /X25,"*     View LOCATION of         *"
    TYPE /X25,"*                              *"
    x = 1
    WHILE (x <= num_regions) DO
      TYPE /X25,"*   ",x,". Work Region #", x,"  *"
      IF (x == num_regions) THEN
        TYPE /X25,"*   ",x+1,". Previous MENU   *"
      END
      x = x+1
    END
    TYPE /X25,"*                              *"
    TYPE /X25,"*******************************"
    TYPE /C1,/X32,"ENTER CHOICE",/S
    PROMPT ": ", loc
    IF (loc > num_regions) GOTO 100  ;Return
    MOVE pic_loc[loc]
    num = 1
    GOTO 20  ;Return to 'LOCATION MENU'
100 TYPE " "
.END
```

A.6 Program Start.up

```
.PROGRAM start.up()
    LOCAL $file, cam.vert, num, $ans, $part

; If the camera is not calibrated, recall calibration data

    IF NOT DEFINED (vtran[1]) THEN
      VGETCAL(1) va[]
      IF va[9] <> 0 THEN
        SET vtran[1] = TRANS(va[9],va[10],0,0,180,va[11])
      ELSE
        $file = "c:armcam1.dat"
        cam.vert = 1
        TYPE /C1,/X5,"Recalling camera calibration data."

; CALL system subroutine 'load.area' to recall data.
```

```
        CALL load.area($file, cam.vert, threshold,
blacklight, vtran[cam.vert], vb[], $error)
      END
    END

; Set desired vision system parameters.

    PARAMETER V.MAX.AREA = 15000
    PARAMETER V.MIN.AREA = 2900
    PARAMETER V.MAX.VER.DIST = 1.9
    PARAMETER V.THRESHOLD = 110
    PARAMETER V.MAX.TIME = 1.1

; Determine the number of prototypes all ready in the system

    num = 1
    VSHOW ^B11000, $protos[num]
    WHILE VFEATURE(1) DO
      num = num+1
      VSHOW ^B1000, $protos[num]  ;array of prototype names.
    END

; The 'VISION PROTOTYPE MENU'...an experimental menu only

 50 TYPE $clear.screen
    TYPE /C2,/X25,"VISION PROTOTYPE MENU",/C!
    TYPE /X30,"1. CIRCLE"
    TYPE /X30,"2. SECTION"
    TYPE /X30,"3. TRIANGLE"
    TYPE /X30,"4. SQUARE"
    TYPE /X30,"5. PENTAGON"
    TYPE /X30,"6. HEXAGON"
    TYPE /X30,"7. TRAPEZOID"
    TYPE /X30,"8. CROSS"
    TYPE /X30,"9. OVAL"
    TYPE /X29,"10. STAR"
    TYPE /X29,"11. CONTINUE"
    TYPE /C1,/X29,"ENTER SELECTION",/S
    PROMPT ": ", ans

; Load the prototype requested from the hard drive.

    CASE ans OF
      VALUE 1:
        $file = "c:circle.vs"
        $part = "CIRCLE."
      VALUE 2:
        $file = "c:section.vs"
        $part = "SECTION."
      VALUE 3:
        $file = "c:triangle.vs"
        $part = "TRIANGLE."
      VALUE 4:
```

```
          $file = "c:square.vs"
          $part = "SQUARE."
        VALUE 5:
          $file = "c:pentagon.vs"
          $part = "PENTAGON."
        VALUE 6:
          $file = "c:hexagon.vs"
          $part = "HEXAGON."
        VALUE 7:
          $file = "c:trapezoid.vs"
          $part = "TRAPEZOID."
        VALUE 8:
          $file = "c:cross.vs"
          $part = "CROSS."
        VALUE 9:
          $file = "c:oval.vs"
          $part = "OVAL."
        VALUE 10:
          $file = "c:star.vs"
          $part = "STAR."
        VALUE 11:
          GOTO 100  ;take a picture and plan
        ANY
          GOTO 50  ;Return to 'VISION PROTOTYPE MENU'
      END
      num = num+2
      IF (num > 6) THEN
        TYPE /C1,/X5,"THE VISION SYSTEM MEMORY CAN HOLD A
MAXIMUM OF SIX PROTOTYPES.",/C1
        PROMPT "     Press RETURN to continue.", $ans
        GOTO 50  ;Return to 'VISION PROTOTYPE MENU'
      END
      TYPE /C1,/X5,"Loading prototype ",$part
      VLOAD (5) $file
      GOTO 50
  100 TYPE /C1, "Taking a picture and planning recognition...
PLEASE WAIT.",/C1
      VDISPLAY 3
      VPICTURE
.END
```

APPENDIX B

LISTING OF VISION HEURISTIC PROGRAM

B.1 Program Sh.look

```
.PROGRAM sh.look()
     TYPE $clear.screen,/C2
     seen = 0

; Memorize initial robot arm location

     HERE temp

; Take an initial picture and record all vision information.

     CALL take.a.picture()
     SET corner = temp:RZ(-jt[4]):vtran[1]
     SET center = corner:TRANS(96.52,76.2,0,0,180,0)
     objnum = 0
     WHILE (VQUEUE(0) > 0) DO
       VLOCATE (0) $part, vloc  ;takes next image off queue
       objnum = objnum+1
       SET objloc[objnum] = picloc:RZ(-jt[4]):vtran[1]:vloc
       $objname[objnum] = $part
       verify[objnum] = VFEATURE(9)
     END

; Reduce camera view area for centering

     PARAMETER V.FIRST.LINE = 100
     PARAMETER V.FIRST.COL = 75
     PARAMETER V.LAST.LINE = 383
     PARAMETER V.LAST.COL = 300
     kount = 1
     WHILE (kount <= objnum) DO
       IF (verify[kount] < 90) THEN
         TYPE /X5,"I'm centering the thing for a better
look..."
         CALL center()
         numpic = 1
 20      CALL take.a.picture()
         numpic = numpic+1
         VLOCATE (0) $part1, vloc1
         IF VFEATURE(1) THEN
           IF (VFEATURE(9) > verify[kount]) THEN
             verify[kount] = VFEATURE(9)
```

```
            $objname[kount] = $part1
            SET objloc[kount] = picloc:RZ(-jt[4]):
vtran[1]:vloc1
            IF (verify[kount] > 90) GOTO 30  ;Jump out
          END
          IF (numpic < 5) GOTO 20  ;Take another picture
        ELSE
          GOTO 500  ;Test next object seen
        END
      END
 30   SET objloc[kount] = SHIFT(objloc[kount] BY 0,0,-187)

; Reject the unrecognized part.

      IF $objname[kount] == "?" THEN
        TYPE /X5,"I didn't recognize that thing.",/C1
        CALL pick_place(objloc[kount],reject_loc)

; Do a database match with the vision information.

      ELSE
        TYPE /X5,"I found the ",$objname[kount]," and I'm
",verify[kount],"% sure.",/C1
        i = 1
        WHILE (i <= part[0,0]) DO

; Find out which hole was seen.

          IF flag == 10 THEN
            j = 1
            WHILE (j <= part[i,0]) DO
              IF $objname[kount] == $hole[i,j] THEN
                SET insert_loc[part[i,j]] =
SHIFT(objloc[kount] BY 0,0,20)
                GOTO 500  ;Test next object seen
              END
              j = j+1
            END

; Find out which component was seen.

          ELSE
            j = 1
            WHILE (j <= part[i,0]) DO
              IF $objname[kount] == $comp[i,j] THEN
                SET comp_loc[part[i,j]] = objloc[kount]
                seen = part[i,j]
                j = part[i,0]  ;Jump out of loop
                i = part[0,0]
              END
              j = j+1
            END
            IF $objname[kount] == $assembly[i] THEN
```

```
                SET base_loc[i] = objloc[kount]
                i = part[0,0]  ;Jump out early
              END
            END
            i = i+1
          END
        END
 500    kount = kount+1
      END

; Enlarge field of view & move robot back before returning.

      PARAMETER V.FIRST.LINE = 1
      PARAMETER V.FIRST.COL = 1
      PARAMETER V.LAST.LINE = 483
      PARAMETER V.LAST.COL = 375
      MOVE temp
      BREAK
.END
```

B.2 Program Take.a.picture

```
.PROGRAM take.a.picture()

; Find joint 4 position in picture location.

      HERE picloc
      HERE #picloc
      DECOMPOSE jt[1] = #picloc
      VPICTURE (1)  ;Take a picture on camera 1.
      VLOCATE (,2) "nothing"  ;Wait until processing complete.
.END
```

B.3 Program Center

```
.PROGRAM center()
      DECOMPOSE a[1] = temp
      DECOMPOSE b[1] = center
      SET trans[1] = TRANS(a[1],a[2],a[3],a[4],a[5],b[6])
      SET trans[2] = INVERSE(center):objloc[kount]
      MOVE trans[1]:trans[2]
      BREAK
.END
```

VITA

Allan Wayne Butler [redacted] His parents are Allan Lloyd Butler and Jeanie Ruth Butler (Jones). After attending the University of Washington (U of W) for a year, Allan served two years as a missionary for the Church of Jesus Christ of Latter-Day Saints in Rome, Italy. Returning home, Allan resumed his studies at the U of W. In 1983 he transferred to Brigham Young University (BYU) in Provo, Utah where he met and married Beverly Dawn Montague on 21 April 1984. In June of 1985, Allan was commissioned a 2nd Lieutenant in the U. S. Air Force, graduated from BYU with a degree in Mechanical Engineering, and became a father to Michael Wayne Butler. Allan attended pilot training at Reese AFB, Texas, and later was assigned as a Contracting Officer at Hill AFB, Utah. In June 1987 David Allan Butler was born. While at Hill AFB, Allan developed many computer applications that were well received by the Air Force. One award he received was the 1988 Commander-in-Chief's Installation Excellence Award given by the Secretary of Defence. In 1988, Allan was selected as one of the "Outstanding Young Men of America". Later he was assigned to the Industrial Engineering Department at Texas A&M University to work on a master's degree. While at Texas A&M, [redacted] was born. Allan is now a Captain in the U. S. Air Force and a member of Alpha Pi Mu Honor Society. Address: 1116 E Toppenish Ave, Toppenish, WA 98948.